Software Engineering Institute

Cloud Computing at the Tactical Edge

Soumya Simanta
Grace A. Lewis
Edwin Morris
Kiryong Ha (Carnegie Mellon University School of Computer Science)
Mahadev Satyanarayanan (Carnegie Mellon University School of Computer Science)

October 2012

TECHNICAL NOTE
CMU/SEI-2012-TN-015

Research, Technology, and System Solutions (RTSS) Program

http://www.sei.cmu.edu

Carnegie Mellon

This material is based upon work funded and supported by the United States Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

This report was prepared for the

SEI Administrative Agent
AFLCMC/PZE
20 Schilling Circle, Bldg 1305, 3rd floor
Hanscom AFB, MA 01731-2125

Table of Contents

List of Figures

List of Tables

Abstract

Handheld mobile technology is reaching first responders, disaster-relief workers, and soldiers in the field to aid in various tasks, such as speech and image recognition, natural-language processing, decision making, and mission planning. However, these applications are computation intensive, so it is necessary to consider that (1) mobile devices offer less computational power than conventional desktop or server computers, (2) computation-intensive tasks consume large amounts of battery power, and (3) networks in hostile environments, such as those experienced by first responders and soldiers in the field, are often unreliable, and bandwidth is limited and inconsistent. While there has been considerable research in code offload to the cloud to enhance computation and battery life, most of this work assumes reliable connectivity between the mobile device and the cloud—an invalid assumption in hostile environments. This technical note presents a reference architecture for mobile devices that exploits cloudlets—virtual-machine-based, code-offload elements—that are in single-hop proximity to the mobile devices that they serve. Two implementations of this reference architecture are presented, along with an analysis of architecture tradeoffs.

1 Introduction

The Department of Defense (DoD) is increasingly interested in having soldiers carry or wear handheld mobile computing devices to support their mission needs [Morris 2011]. Potentially, front-line soldiers can use handheld devices for help with tasks such as speech and image recognition, natural-language processing, decision making, and mission planning. However, several obstacles impede achieving the capabilities that soldiers need when carrying handheld computing devices in mission environments. First, mobile devices offer less computational power than conventional desktop or server computers, and are therefore not an ideal computation platform for tasks such as natural-language processing and complex decision making. Second, computation-intensive tasks, such as image recognition or even global positioning systems (GPS), consume large amounts of battery power. Third, networks in mission environments are often unreliable, and bandwidth is limited and inconsistent [Satyanarayanan 1996]. To refer to these mission environments in which front-line soldiers must operate, we use the phrase *tactical edge.*[1]

A technique for overcoming some of these challenges is known as *cyber-foraging*—the leveraging of external resources to augment the capabilities of resource-limited mobile devices [Balan 2002, 2007; de Lara 2001; Flinn 2001, 2002; Goyal 2004; Kemp 2009; Ok 2007; Satyanarayanan 2001]. However, related work on offloading computation to conserve battery power in mobile devices relies on (1) the conventional internet or (2) environments that tightly couple applications running on handheld devices and servers on which computations are offloaded (see Section 6). While cyber-foraging conserves battery power and reduces computing power requirements, it does not address the challenges of unreliable networks and dynamic environments.

This technical note presents a strategy to overcome the challenges of cyber-foraging for mobile platforms at the tactical edge by using *cloudlets*—discoverable, localized, stateless servers running one or more virtual machines (VMs) on which soldiers can offload resource-intensive computations from their handheld mobile devices. Cloudlets enhance processing capacity and conserve battery power while at the same time providing ease of deployment in the field.

[1] First responders operating in crisis situations such as earthquakes, tsunamis, and other similar emergencies can also be considered to be operating at the tactical edge.

2 Cloudlets as Intermediate Offload Elements

Code offload from mobile devices to cloud environments has been the topic of much recent research [Christensen 2009; de Leusse 2008; Kumar 2010; Li 2009; Marinelli 2009; Palmer 2009; Zhang 2009]. Applications such as Siri also use this approach for voice recognition [Dilger 2011]. However, an underlying assumption in these approaches is connectivity to the cloud, which is not always available or reliable at the tactical edge.

A high-level architecture for code offload in hostile environments—such as the tactical edge—is proposed by Ha and colleagues and presented in Figure 1 [Ha 2011]. This architecture inserts an intermediate layer between the central core (i.e., enterprise cloud) and the mobile devices. The large, centralized core at the heart of the architecture is the focus of system administration. This core could be implemented as one of Amazon's data centers or a private enterprise cloud located in a stable and secure environment, far from physical threats. At the edges of this architecture are offload elements for mobile devices. These elements, or cloudlets, are dispersed, and each is located close to the mobile devices that it serves [Satyanarayanan 2009]. This architecture decreases latency by using a single-hop network and potentially lowering battery consumption by using WiFi or short-range instead of broadband wireless, which typically consumes more energy [Lehr 2002; Cuervo 2010].

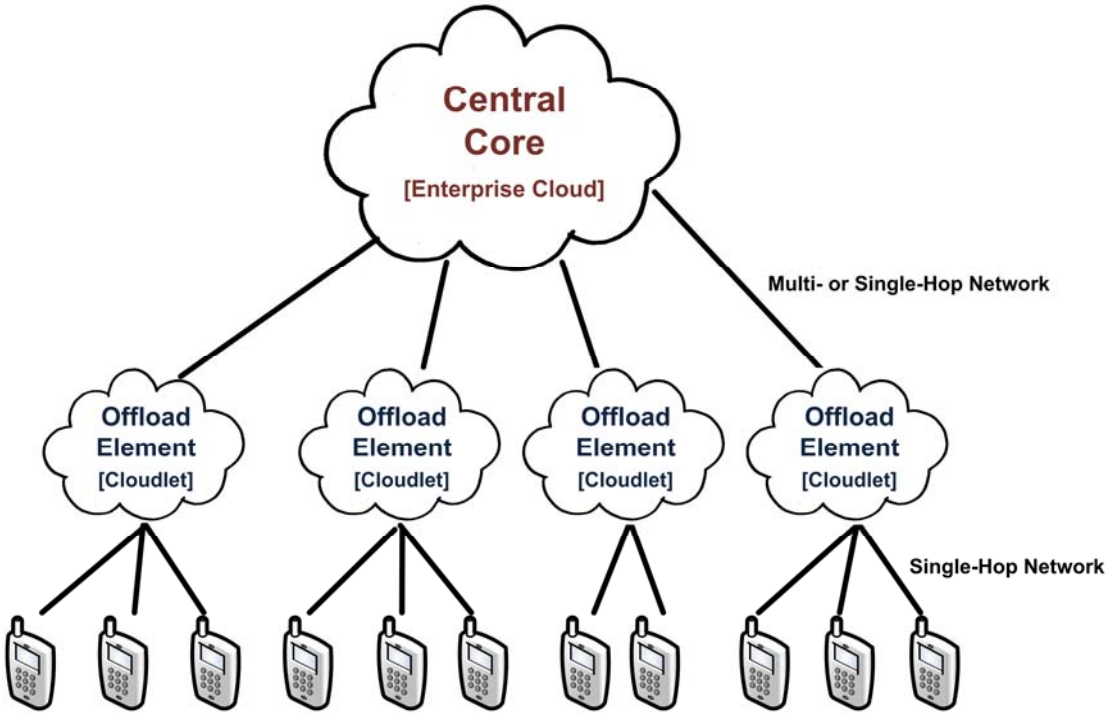

Figure 1: Architecture for Code Offload

A key attribute of this architecture is that the offload elements are stateless. They may cache state persistently on their local storage, but the provisioning of the cache is done from the central core.

After installation, an offload element self-provisions from the central core. A final step of provisioning can occur just before use by a mobile device.

This architecture is resilient to wide-area-network failures. A mobile device does not need to communicate with the core during an offload operation; it needs to communicate only with its physically closest offload element. The mobile device can continue offloading operations even when its offload element is totally disconnected from the core. Communication between the offload element and the core is necessary only during provisioning. Once an offload element is provisioned, its connection to the core can be disrupted without affecting offload service to mobile clients. Adding a new offload element or replacing an existing one involves very little setup or configuration effort.

One approach to offloading code to cloudlets is VM synthesis [Wolbach 2008; Satyanarayanan 2009]. This approach involves offloading an application overlay from the mobile device to a cloudlet. An application overlay represents the difference between a Base VM with only the operating system installed and a VM with the application installed. As long as the Base VM Image and associated VM manager are available on the offload element, the application can be made to execute even if it was not previously deployed to that specific platform. In effect, the mobile device becomes the vector by which the needed application is deployed to the tactical-edge environment. This approach takes advantage of properties of VMs that reduce hardware dependencies—the same VMs can operate on several hardware platforms, and a single hardware platform can support multiple VMs—to provide flexibility in highly volatile tactical-edge environments, where it is difficult to assure the existence of the right hardware or OS platform for cyber-foraging.

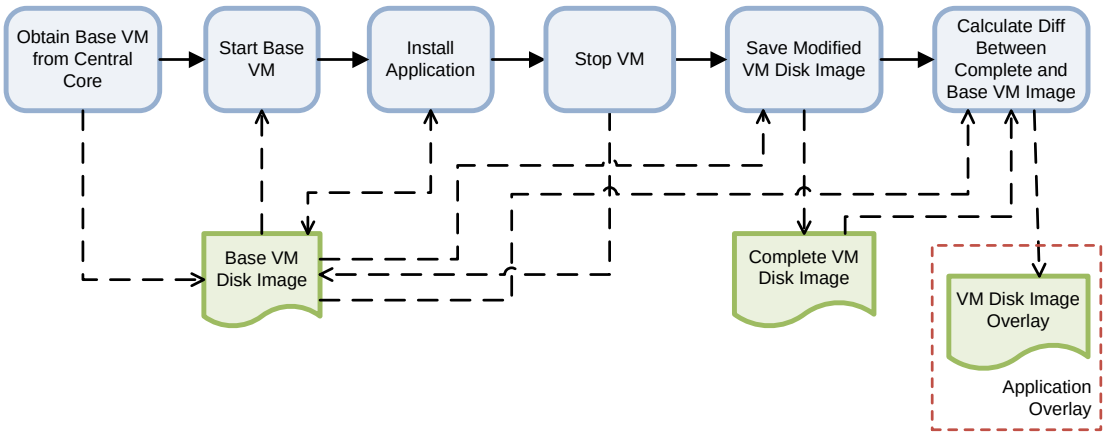

Figure 2: Application Overlay Process

An application overlay is created once per application as Figure 2 depicts. Here the Base VM is a VM disk image file that is obtained from the central core and saved to a cloudlet that runs a VM manager compatible with the Base VM. The VM manager starts the Base VM and the application is installed. After installation, the VM is shut down. At this point, a copy of the modified Base VM Disk Image is saved as the Complete VM Disk Image. The application overlay is calculated as the binary diff [2] between the Complete VM Disk Image and the Base VM Disk Image. The

[2] The binary diff tool uses the VCDIFF (RFC 3284) format and compression.

Base VM is then deployed to any platform that might conceivably serve as a cloudlet. As we previously suggested, the cloudlet may support multiple VMs, thereby reducing hardware dependencies on the offload element and between the offload element and mobile device. The mobile device would then carry as many overlays as necessary for achieving a mission and would be able to execute on any cloudlet that has the same Base VM with which the application overlay was created.

The cloudlet application of VM technology provides great flexibility in the supported types and platforms of applications and also reduces setup and administration time. For example, a cloudlet might run in a tactical operations center (TOC) or a Humvee. Because no application-specific software is installed on the server, there is no need to synchronize release cycles between the client and server portions of apps, which greatly simplifies the deployment and configuration management of apps in the field.

VM synthesis is particularly useful in tactical environments characterized by unreliable networks and bandwidth, unplanned loss of cyber-foraging platforms, and a need for rapid deployment. For example, imagine a scenario in which a soldier must execute a computation-intensive app configured to work with cloudlets. At runtime, the app discovers a nearby cloudlet located on a Humvee and offloads the computation-intensive portion of code to it. However, due to enemy attacks, loss of network connectivity, or exhaustion of energy sources on the cloudlet, the mobile app is disconnected from the cloudlet. The mobile app can then locate a different cloudlet (e.g., in an unmanned aerial vehicle [UAV]) and have the app running in a short amount of time with no need for any configuration on the app or the cloudlet. This runtime flexibility enables the use of whatever resources become opportunistically available as well as replacement of lost cyber-foraging resources and dynamic customization of newly acquired cyber-foraging resources.

The following sections present the reference architecture for code offload in hostile environments and details of two prototype implementations.

3 Reference Architecture

Tactical environments are characterized by uncertainty in available resources such as computational capability and bandwidth. In addition, many applications that are useful in these environments are computation intensive; these include face recognition, natural-language processing, route calculation, and text recognition, all of which require some form of input from sensors on the device. Soldiers and first responders executing missions are often away from their bases for many hours and cannot afford to carry many extra batteries. Therefore, a solution for code offload in hostile environments must consider (1) native apps that exploit device sensors, (2) code offload elements that can be quickly configured and deployed, and (3) battery consumption on the mobile device.

Figure 3 presents a reference architecture for mobile devices that exploit cloudlets for code offload. The major components of this architecture are the Cloudlet Host and the Mobile Client.

The *Cloudlet Host* is a physical server that hosts the following:

- a discovery service that broadcasts the cloudlet IP address and port so that it can be located by mobile devices
- the Base VM Image that is used for VM synthesis
- a Cloudlet Server that handles code offload in the form of application overlays, performs VM synthesis, and starts guest VM instances with the resulting VM images
- a VM Manager that serves as a host for all guest VM instances that contain the computation-intensive server component of the corresponding mobile app

The *Mobile Client* is a handheld or wearable device that hosts the following:

- a Cloudlet Client app that discovers cloudlets and uploads application overlays to the selected cloudlet
- a set of Cloudlet-Ready Apps that operate as clients of the server code running in the cloudlet

The Mobile Client stores an application overlay for each Cloudlet-Ready App that a user would conceivably want to execute and for which computation offloading is appropriate. Each application overlay is generated from the same Base VM Image that resides in the cloudlet. In an operational setting, these Base VM Images could be retrieved from the central core shown in Figure 1.

To validate the feasibility of the proposed reference architecture for tactical environments, we constructed a prototype that we describe in the next section.

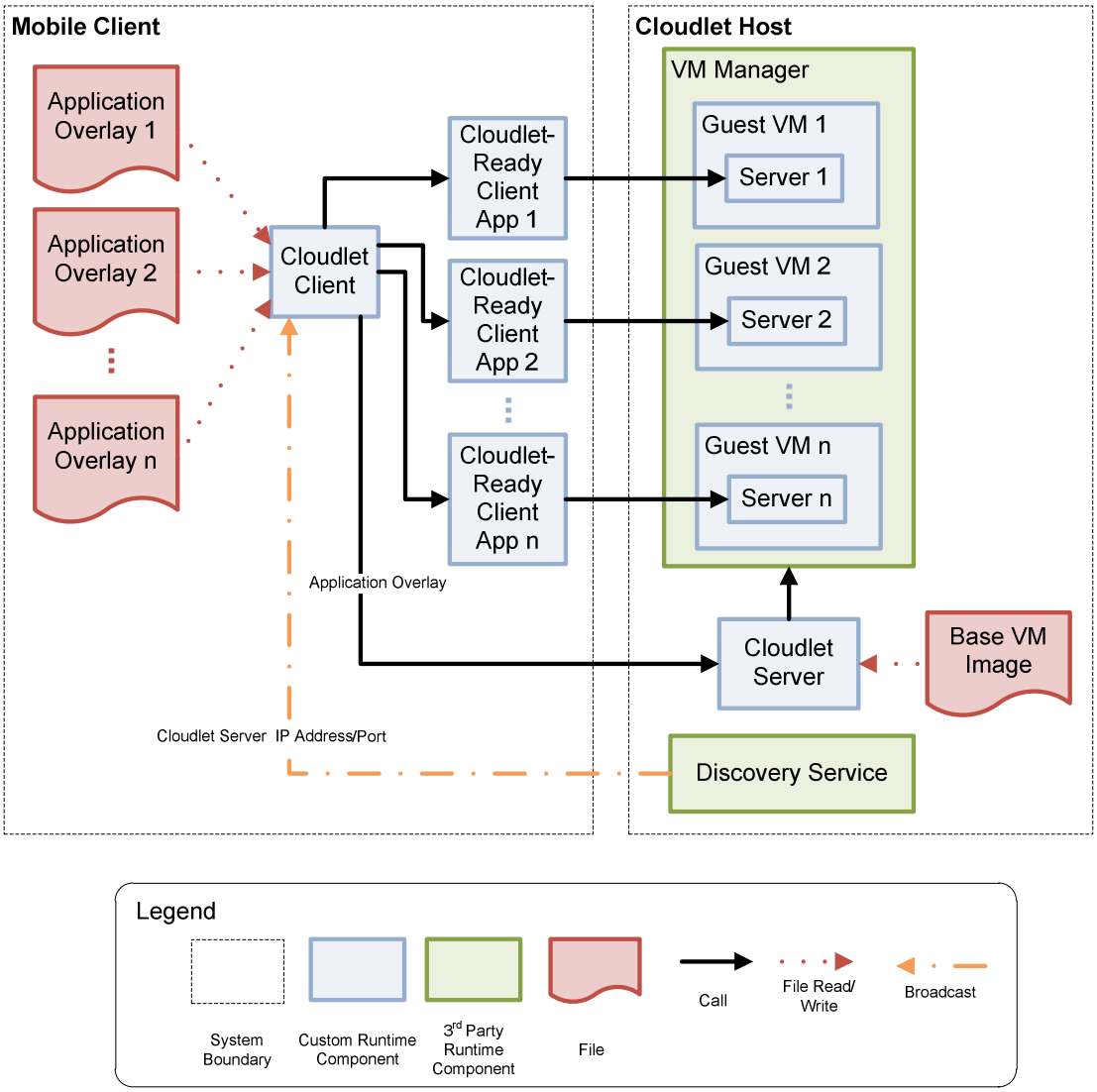

Figure 3: A High-Level Reference Architecture for Cloudlet-Based Code Offload

4 Initial Prototype

The initial prototype is an implementation of a face-recognition application in which the client is an Android app and the cloudlet contains the computation-intensive code that performs face recognition. In summary, the client

1. locates a cloudlet via a discovery protocol
2. sends the application overlay (Face-Recognition Server code) to the cloudlet for VM synthesis
3. captures images and sends them to the Face-Recognition Server that now runs in the cloudlet

This initial prototype was created based on the reference architecture that Figure 3 illustrates. Figure 4 depicts the prototype architecture.

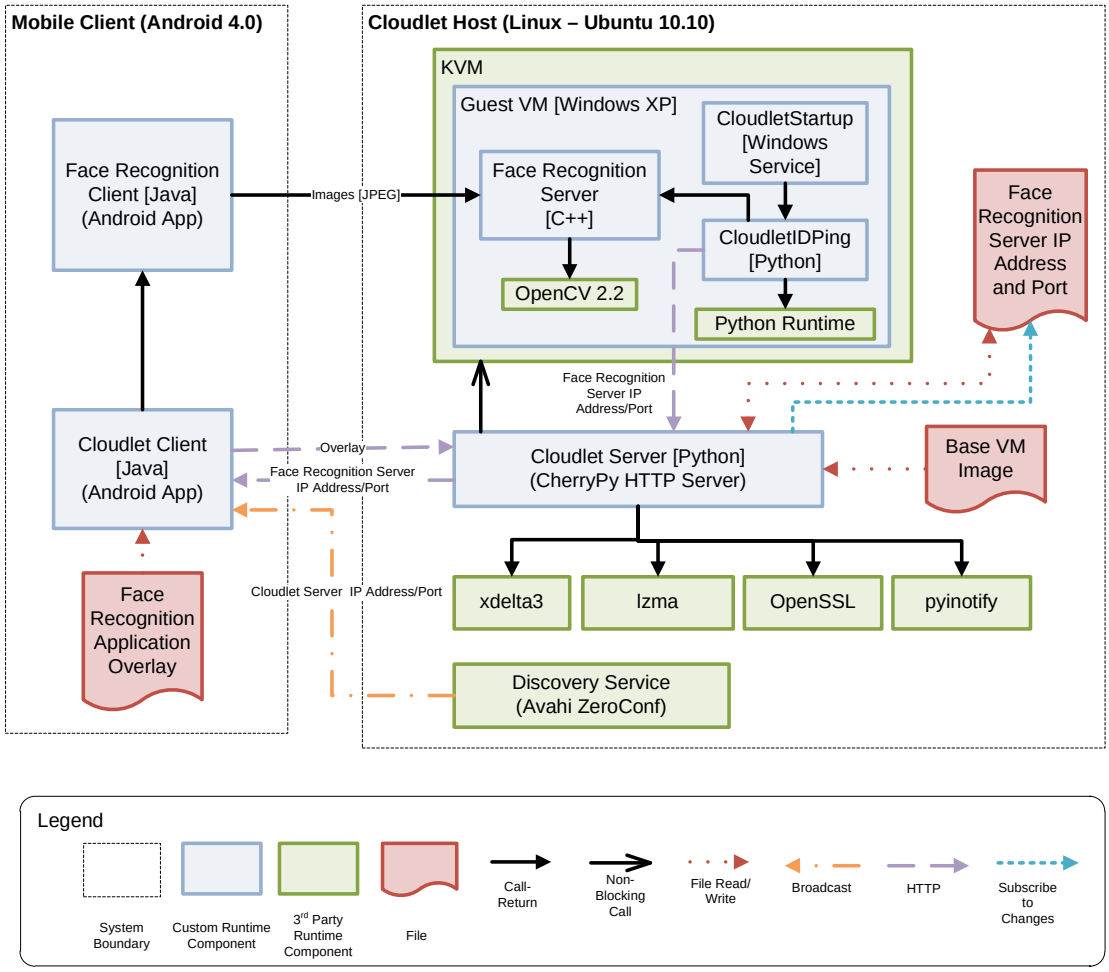

Figure 4: The Architecture of the Initial Prototype

4.1 Base VM Image Creation

The Base VM Image for this prototype is a 3GB image with Microsoft XP plus the necessary system updates. To keep the size of the image small, "system restore" was turned off and unnecessary system components were removed using the DiskCleanup utility. However, additional components were included in the Base VM Image to enable communication between the Guest VM and the Cloudlet Server. This additional complexity was one of the reasons for revising the initial prototype (see Section 5.1).

A major difference between the prototype and the cloudlet work presented by Satyanaranyan is the type of client involved [Satyanaranyan 2009]. Satyanaranyan's work uses a virtual network computing (VNC) client that acts as a remote desktop for the VM [Richardson 1998]. In our prototype, the Face Recognition Client is a rich client (native app) that interacts with a Face Recognition Server inside the VM (i.e., a server within a server). The Face Recognition Client requires the IP address and port of the Face Recognition Server to establish a network connection. However, the Cloudlet Server does not know directly the IP address and port of the synthesized VM, because the IP address is assigned by the Dynamic Host Configuration Protocol (DHCP) server executing in bridged mode, and the VM host has no visibility into that assignment. The Cloudlet Client uses an HTTP request to start the cloudlet setup and expects an HTTP response from the Cloudlet Server that contains the IP address and port of the Face Recognition Server.

To solve this problem, a CloudletStartup Windows service (implemented in Python) is included in the Guest VM that performs three functions:

1. starts the Face Recognition Server—in a separate thread
2. reads the IP address and port from the Face Recognition Server configuration file and communicates it to the Cloudlet Server in an HTTP POST
3. sends a periodic heartbeat to the Cloudlet Server in an HTTP POST to indicate that it is still running

In addition, the Windows registry entry KeepAliveTime was changed to one second (default is two hours) so that the Face Recognition Server can close the socket if the communication channel between the Face Recognition Client and the Face Recognition Server is broken [Microsoft 2011]. The communication channel can break when there is (1) a phone disconnection from the network (2) an error in the Face Recognition Client that causes it to exit without closing the socket, or (3) a network issue in the VM.

After completion of these steps, the VM was shut down and the resulting image file was saved as the Base VM Image.

4.2 Tools for Overlay Creation

The overlay for the face-recognition application was created by following the steps presented in Figure 2. The VM Manager was started using the Base VM Image created as we described in the previous section, the Face Recognition Server was installed, and the VM shut down. The specific tools to "Calculate Diff between Complete and Base VM Image" are

- xdelta3: an open-source binary diff tool that generates a file as the difference between the Base VM Disk Image and the Complete VM Disk Image [xdelta 2012]

- lzma: a set of public-domain libraries and tools that compress the file from the previous step using lzma—the Lempel-Ziv-Markov chain data-compression algorithm [7Zip 2012]

The resulting file is the application overlay. The Cloudlet Server uses these same tools in reverse order when it performs VM synthesis.

4.3 Cloudlet Host

The Cloudlet Host is an Ubuntu 10.10 Linux server that hosts the following subcomponents:

4.3.1 KVM (Kernel-Based Virtual Machine)

The virtualization infrastructure that we used in the prototype is KVM [KVM 2012a]. We made this choice primarily for ease of implementation; KVM is open source and has good community support. KVM runs a Guest VM for each offloaded application. The Guest VM for the face-recognition application is Windows XP, but KVM supports most popular operating systems [KVM 2012b].

4.3.2 Cloudlet Server

The core of the Cloudlet Server is an HTTP Server implemented using CherryPy, an open source Python implementation of a basic HTTP server [CherryPy 2012]. The Cloudlet Client sends the compressed overlay as an HTTP POST request. Upon receipt of the overlay, the following operations execute:

1. The overlay is decompressed using the tools listed in Section 4.2.
2. VM synthesis is performed through use of xdelta3 to apply the overlay to the Base VM Image and create the VM Image that contains the Face Recognition Server.
3. The synthesized VM is started in bridged network mode (i.e., it obtains a unique IP from the DHCP server), so that the Guest VM has a unique network-accessible IP address, and waits for notification that the Guest VM has successfully started [Ubuntu 2012a]. The Cloudlet Server uses pyinotify—a Python file-system change-monitoring utility—to subscribe to changes in the face-recognition server IP address and port file [Github 2012]. This provides a mechanism for inter-thread communication.
4. On startup, the Guest VM executes the CloudletStartup Windows service (see Section 4.1) to communicate the IP address and port back to the Cloudlet Server, where another thread writes the Face Recognition Server IP address and port to a file.
5. As soon as the file changes, the waiting thread in Step 3 receives notification. It reads the file and sends a response to the Cloudlet Client that contains the IP address and port on which the Face Recognition Server will be listening.

4.3.3 Discovery Service

The Discovery Service is based on the Avahi implementation of Zero Configuration Networking (ZeroConf) [Avahi 2012]. Zeroconf is a local network-discovery protocol for creating an IP network [Zeroconf 2012]. The Discovery Service broadcasts the IP address and port of the Cloudlet Server.

4.4 Cloudlet Client

The Cloudlet Client is an Android app that performs the following operations:

1. discovers cloudlets in the local network through information broadcast by the Discovery Service that resides in the Cloudlet Host
2. creates an HTTP connection to the Cloudlet Server for overlay transmission by using the IP address and uploads the overlay
3. waits for an HTTP response from the Cloudlet Server that contains the IP address and port on which the Face Recognition Server is listening
4. writes the IP address and port of the Face Recognition Server to a local file that the Face Recognition Client will use
5. launches the Face Recognition Client

4.5 Face Recognition Client

The Face Recognition Client is an Android app that executes in client/server mode with the Face Recognition Server running in the Guest VM. On launch, the Face Recognition Client reads the local file that contains the IP address and port of the Face Recognition Server and opens a TCP/IP connection to it. Images that the camera captures on the mobile device are sent to the Face Recognition Server.

4.6 Face Recognition Server

The Face Recognition Server is a program written in C++ that uses the OpenCV image-recognition library to process images sent from the Face Recognition Client for training or recognition purposes [OpenCV 2012]. When in recognition mode, it returns coordinates for the recognized faces plus a measure of confidence.

4.7 Prototype Execution

The sequence diagram in Figure 5 represents the initial part of the prototype execution that launches the Guest VM. This part of the execution is independent of the application. At the end of this sequence, the Cloudlet Client is blocked until it receives an HTTP response from the Cloudlet Server.

The sequence diagram in Figure 6 shows the second part of the prototype execution in which the Guest VM is launched and the Face Recognition Server is started. An HTTP response is sent back from the Cloudlet Server to the Cloudlet Client indicating that the Face Recognition Server is running and listening at the returned IP address and port. After receiving this response, the Cloudlet Client starts the Face Recognition Client, which opens a socket to the Face Recognition Server and starts execution.

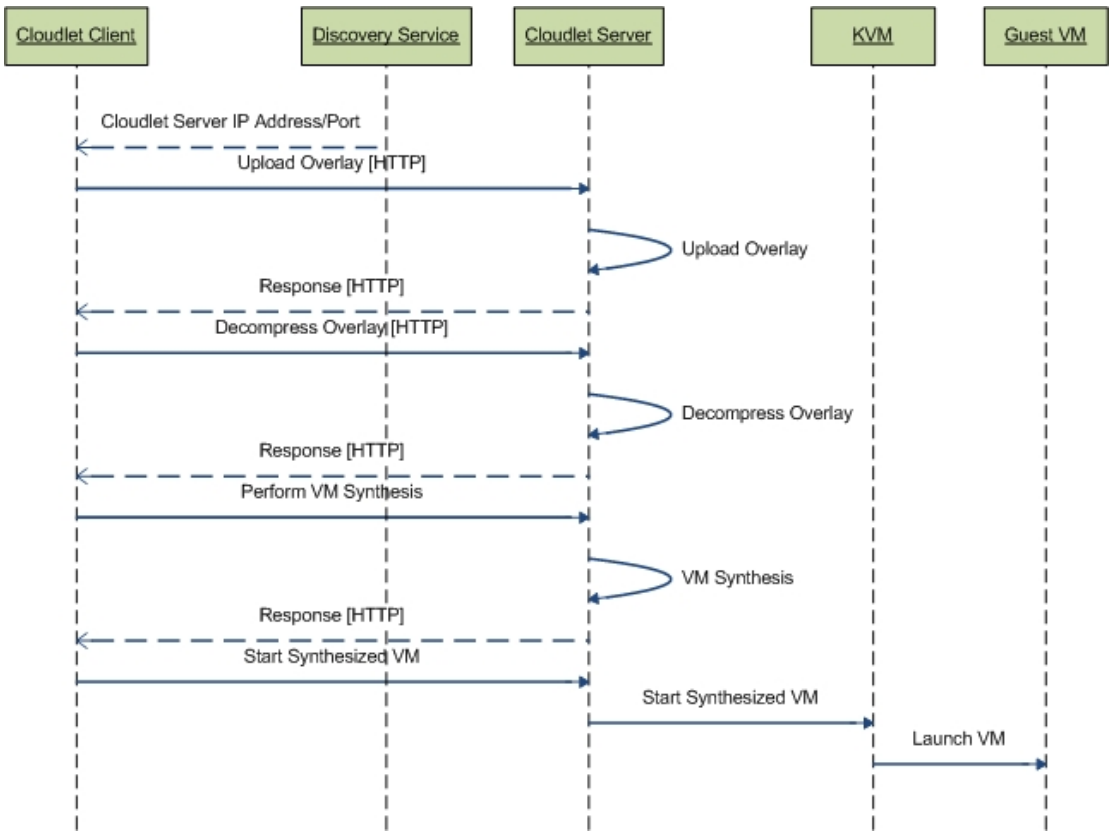

Figure 5: Initial Communication—Cloudlet Discovery to Guest VM Launch for the Initial Prototype

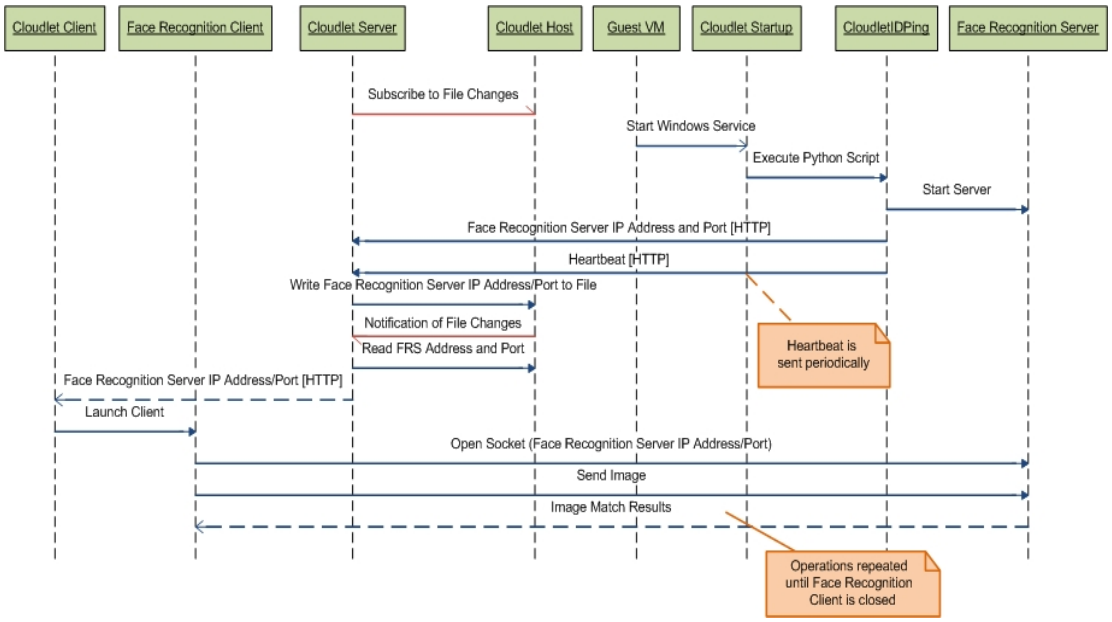

Figure 6: Face Recognition—from Guest VM Launch to App Execution for the Initial Prototype

4.8 Prototype Evaluation

We evaluated the prototype using the previously described face-recognition application and two additional computation-intensive applications that are representative of capabilities that soldiers need at the tactical edge [Ha 2011]. Each application has a client portion that is an Android app and a server portion that corresponds to the computation-intensive offloaded code that runs in the cloudlet. We describe the server-side applications below.

- OBJECT: Linux C++ application based on the CMU MOPED object-recognition libraries [CMU 2012a]. It receives an input image from the mobile client and returns the identities of recognized objects in the image.
- FACE: Windows XP C++ face-recognition application described in Section 4.6
- SPEECH: Windows XP Java application based on the CMU Sphinx-4 Speech Recognition Toolkit [CMU 2012b]. It receives a WAV file from the mobile client and returns a text transcript of the input file.

In addition, an overlay corresponding to a NULL application (VM simply started and stopped) serves as a baseline for the analysis of transmission overhead and battery consumption. Table 1 displays the application data.

Table 1: Application Data for Initial Prototype

Application	Platform	Language	Application Size (MB)	Base VM Disk Image Size (MB)	VM Disk Image Overlay Size (MB)
OBJECT	Linux	C++	27.50	3546	165.32
FACE	Windows XP	C++	17.65	3073	43.55
SPEECH	Linux	Java	51.04	3546	176.23
NULL	Linux	N/A	N/A	3546	0.12

We conducted all experiments using the configuration shown in Figure 7. We measured energy usage with a power monitor from Monsoon Solutions, Inc., and the corresponding power-tool software [Monsoon 2012]. To ensure good experimental control, interactive inputs were scripted.

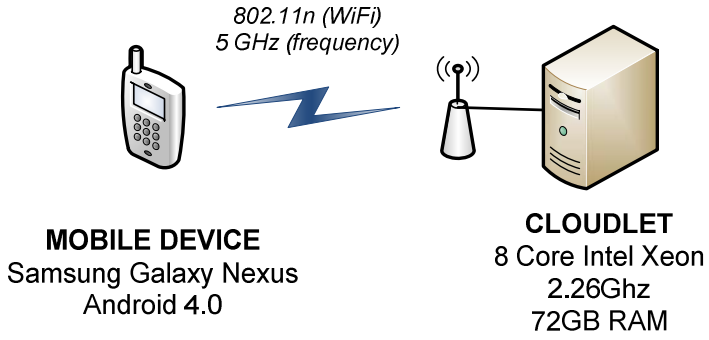

Figure 7: Evaluation Infrastructure Setup

Table 2 lists the average times for each step of the process and the average energy consumption of each application, and Figure 8 displays graphs of the data. All times are measured from the client perspective and include HTTP request/response times.

Table 2: Execution Data for Initial Prototype

Application	Upload Overlay (seconds)	Decompress Overlay (seconds)	VM Synthesis (seconds)	Start VM (seconds)	Energy (Joules)
OBJECT	75.95	17.91	37.24	34.81	189.76
FACE	19.25	4.77	21.91	55.01	82.94
SPEECH	79.89	19.08	35.34	34.95	197.37
NULL	0.23	0.07	34.95	33.97	56.45

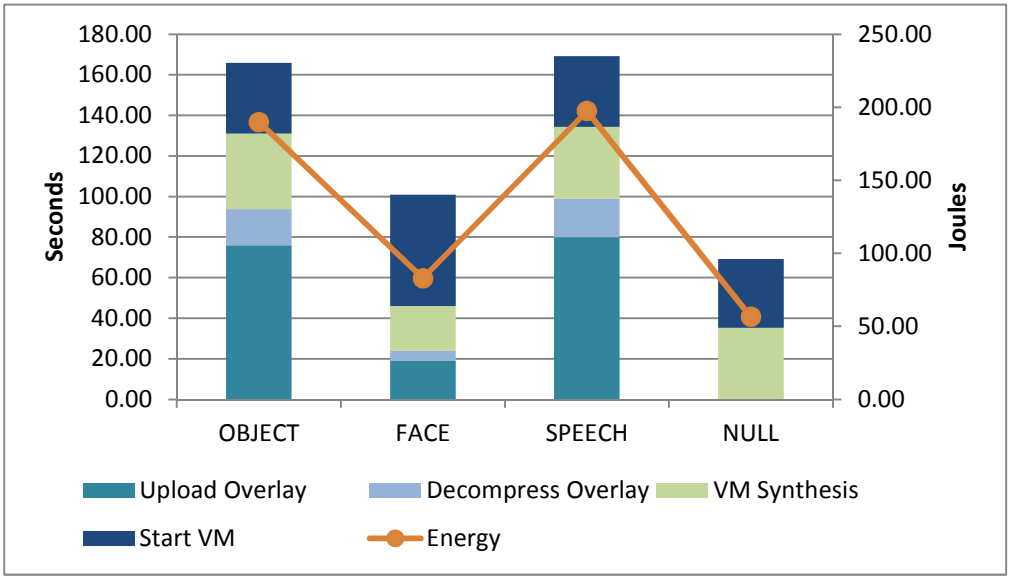

Figure 8: Measures per Application for the Initial Prototype

Table 2 and Figure 8 show that the Upload Overlay operation consumes the largest amount of time, which depends on overlay size. VM Synthesis and Start VM almost equally consume the second-largest time. VM Synthesis time is smaller for FACE because the sum of the base VM size and overlay size is smaller. FACE is also the outlier for Start VM because it is the only application that runs on Windows and therefore has a longer boot time. Energy consumption also depends largely on overlay size. Average application ready time (time between upload overlay and Start VM) is between 101 and 166 seconds for the non-null applications. Given the dependence of these numbers on base VM image size and application overlay size, reducing the size of these files would reduce both application ready time and energy consumption. We addressed file size and implementation complexity in a revised prototype that we describe in the next section.

5 Revised Prototype

In revising the initial prototype, our goals were first, to reduce application complexity and dependencies, and second, to decrease overall application ready time. The revised architecture shown in Figure 9 uses the same implementation of face recognition as for the initial prototype.

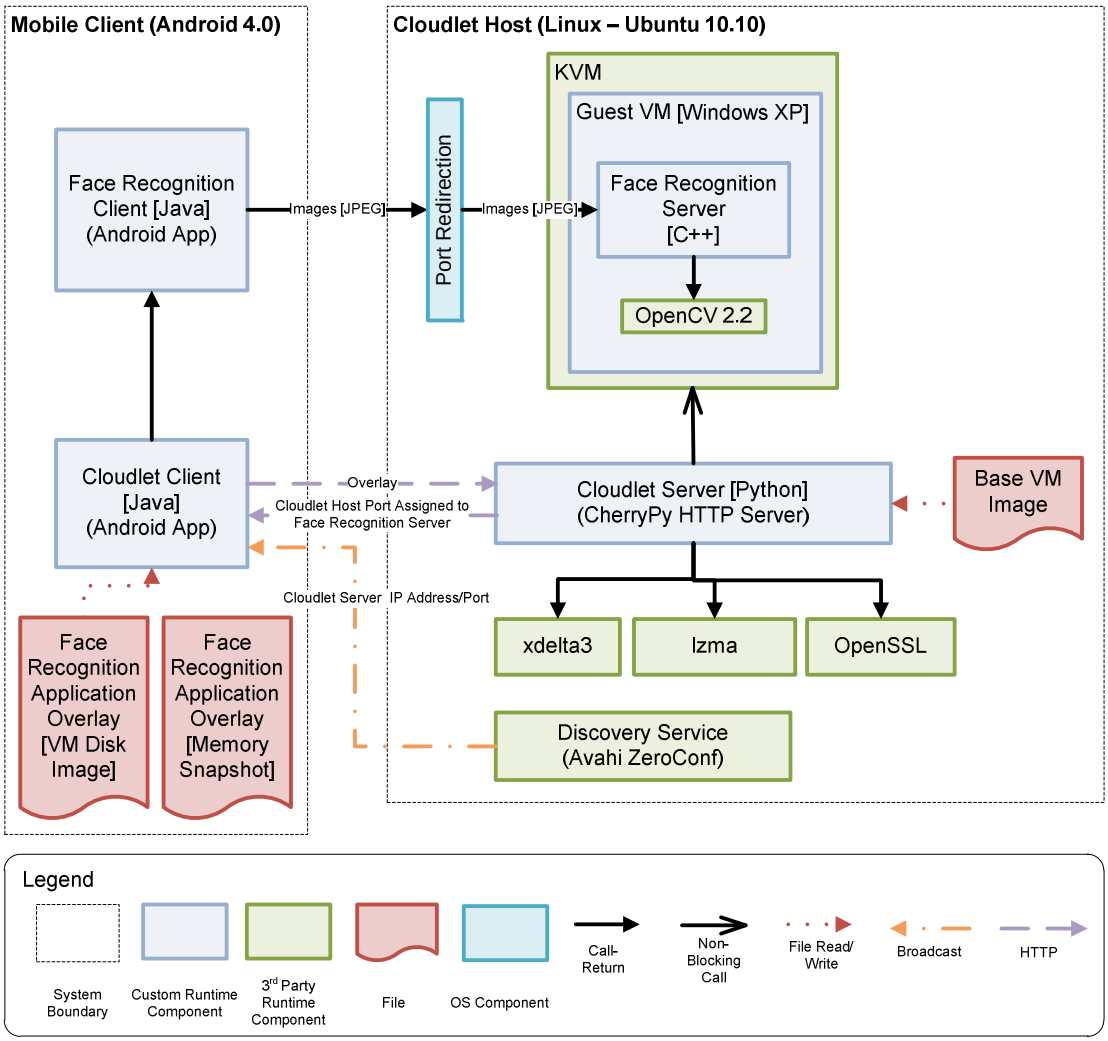

Figure 9: Revised Prototype Architecture

5.1 Main Changes

To address several limitations of the initial prototype, we made the changes described below.

5.1.1 Disk Image Format

The default format for KVM disk images is raw images. Storage for raw images is allocated during creation of the image. For example, if a 3GB image is created, the VM manager writes data inside the image file as needed, and the file always remains this same size—there is no further

storage allocation, regardless of application demand. In essence, the raw image is treated as a disk drive, which tends to result in large images because all storage is pre-allocated as part of the image. This image-size issue is why one challenge for the initial prototype was to create the smallest image possible so that the overlay would be of the smallest size possible as well. Raw images tend to have better performance because there is no need to allocate disk space at runtime.

Another disk-image format that KVM supports is QEMU copy on write 2 (qcow2). The advantage of qcow2 for the prototype is that storage allocation is delayed until actually needed. This means that the size of the image will correspond to the operating system plus the application and, therefore, be optimized, as will be the size of the overlay. A disadvantage of qcow2 is the overhead that storage allocation causes at runtime. Reduction in the amount of time for overlay transition made this tradeoff acceptable for the revised prototype.

5.1.2 Memory Snapshot Overlay Plus Disk-Image Overlay

The initial prototype transfers only the disk-image overlay. This means that the VM is always cold started and requires application-specific scripts to start the application and send connection information back to the client, as explained in Section 4.1.

In the revised prototype, when the base VM image is created, a disk and a memory snapshot are immediately created as well. The three files (Base VM Disk Image, Base Memory Snapshot, and Base Disk Snapshot) now constitute the base VM image. When the base VM image is started, as Figure 10 illustrates, Base Memory Snapshot and Base Disk Snapshot are immediately applied. The application is installed, the VM is suspended, and a second set of snapshots is saved. Overlays are created as the difference between the sets of snapshots, which accounts for a very small set of overlays. The tradeoff is that, in this implementation, two overlays are sent from the client to the cloudlet. However, the advantage is that there is no need to create startup scripts, leading to fewer dependencies on the base VM. In addition, because the VM starts from a suspended state instead of from a stopped state, the VM Start time is faster.

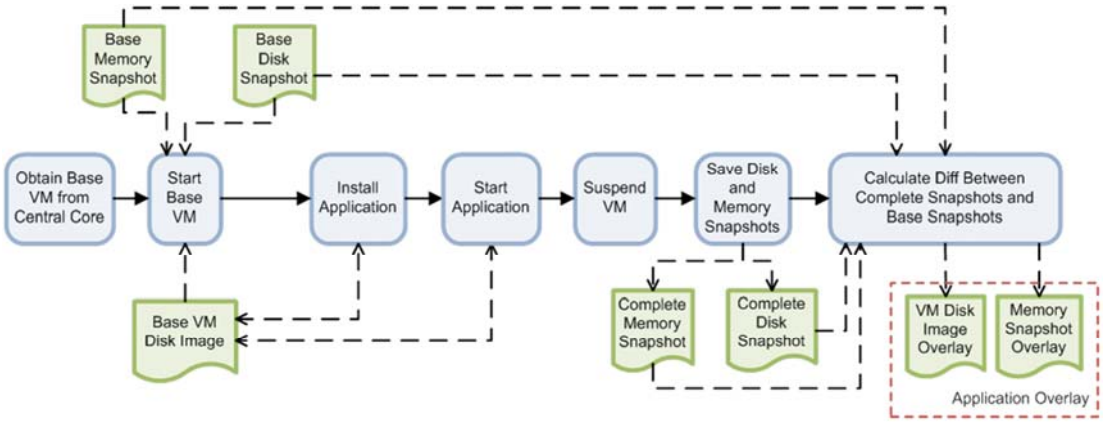

Figure 10: Revised Application Overlay Creation Process

5.1.3 KVM in NAT Mode and Port Redirection

One of the main complexities of the initial prototype was the KVM-to-Cloudlet-Host communication, explained in Section 4.1, that allowed the mobile device to connect directly to the Face Recognition Server inside the VM.

The revised prototype starts the synthesized VM in NAT (Network Address Translation) mode instead of bridged mode, so that the Guest VM can share the same IP address as the Cloudlet Host. However, in NAT mode the Guest VM is not directly accessible from the client. When the Cloudlet Server starts the synthesized VM in NAT mode, it includes the port that the Guest VM should listen on as a parameter. The Cloudlet Server then maps an externally accessible port on the Cloudlet Host to the port assigned to the Guest VM. The externally accessible port number is sent to the client and the Cloudlet Host redirects all communication to the Guest VM. The tradeoff is that NAT is restricted to certain protocols (HTTP, SMTP, FTP) and cannot be bound to ports numbered lower than 1024 without root privileges.

Scalability could become an issue if the port request exceeds the number of available ports. However, our revision greatly simplifies deployment because the Windows service, Python runtime, CloudletIDPing Python script, and pyinotify are no longer necessary. In addition, NAT is more secure than bridged mode because the VM is firewalled from the outside.

5.2 Revised Prototype Execution

The sequence diagram in Figure 11 represents the complete revised prototype execution. Discovery, upload, and decompression are conducted as in the initial prototype. VM synthesis in the revised prototype applies both the disk overlay and the memory snapshot to the Base VM image. The Cloudlet Server starts and launches the synthesized VM on a specific port. This port number is sent back to the Cloudlet Client in an HTTP response. Upon receiving the port number, the Cloudlet Client starts the Face Recognition Client, which opens a socket to the Face Recognition Server and starts execution. The main difference from the initial prototype is that no startup scripts are involved on the cloudlet side.

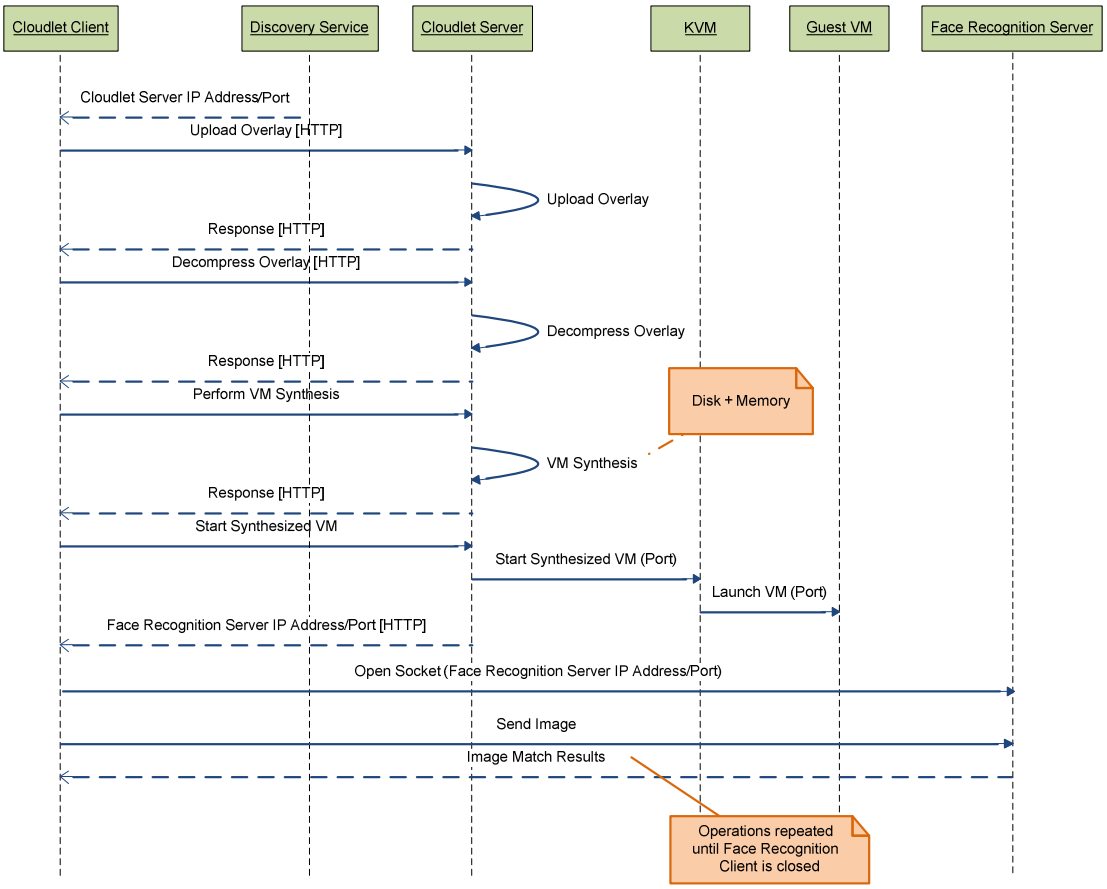

Figure 11: Revised Prototype Execution

5.3 Revised Prototype Evaluation

We evaluated the revised prototype using the same applications as were listed in Section 4.8. Table 3 presents the application data. Overlay size is considerably smaller, except for FACE, due to application characteristics such as language, use of DLLs (dynamic link libraries), configuration files, and so on.

Table 3: Application Data for Revised Prototype

Application	Base VM Disk Image qcow2 (MB)	Base Disk Snapshot qcow2 (MB)	Base Memory Snapshot (MB)	Compressed VM Disk Image Overlay (MB)	Compressed Memory Snapshot Overlay (MB)
OBJECT	3558	17	554	94	293
FACE	2421	15	278	71	101
SPEECH	3558	17	554	86	257
NULL	3558	17	554	1	3

Average times for each step of the process and average energy consumption are shown in Table 4 and graphed in Figure 12.

Table 4: Execution Data for Revised Prototype

Application	Upload Overlays (seconds)	Decompress Overlays (seconds)	VM Synthesis (seconds)	Start VM (seconds)	Energy (Joules)
OBJECT	198.24	43.85	19.97	3.74	357.98
FACE	88.21	19.42	8.10	3.41	158.36
SPEECH	181.53	40.08	18.60	3.61	243.82
NULL	1.04	0.24	5.81	3.46	10.55

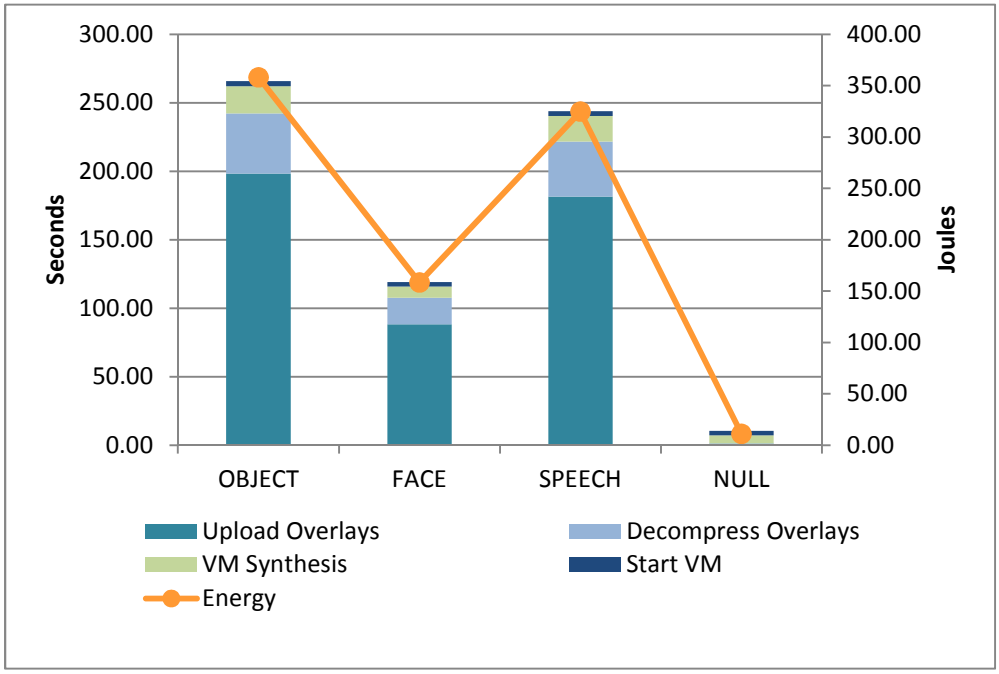

Figure 12: Measures per Application for Revised Prototype

As Figure 13 illustrates, although the reductions in VM Synthesis and Start VM times are considerable with respect to the initial prototype, Upload Overlay and Decompress Overlay times are higher because of the total size of the combined overlays (disk plus memory). Consequently, as Figure 14 shows, energy consumption increases with overlay size. However, implementation, application configuration, and VM-host-guest communication are simplified, and Start VM time is more consistent, regardless of the operating system running inside the VM.

The bandwidth between the mobile device and the cloudlet during the experiments was approximately 13 Mbps, even when using 802.11n wireless. This lower-than-expected data rate may be caused by radio interference in the environment where the experiments were conducted. The bottom line is that, for the revised prototype to pay off, the efficiencies gained in VM Synthesis and Start VM would require supplementation with greater bandwidth.

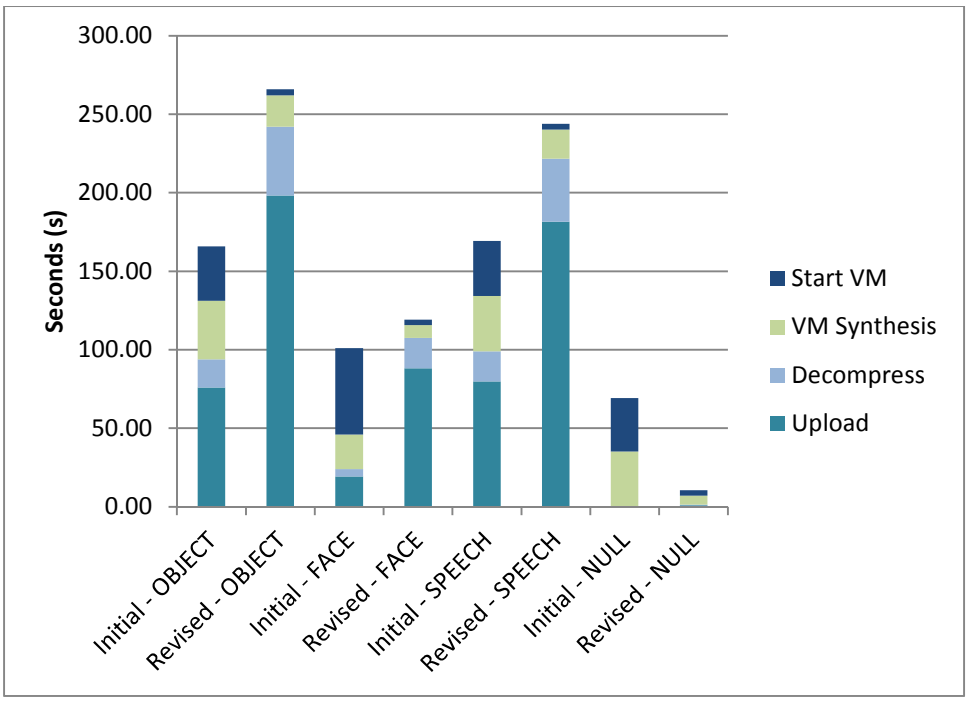

Figure 13: Comparative Execution Data for Initial and Revised Prototypes

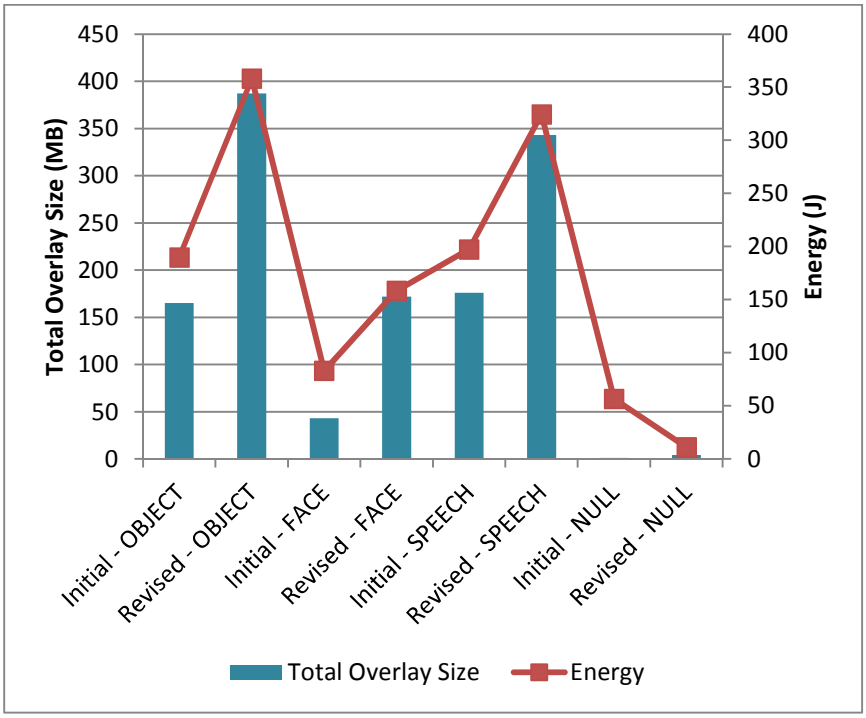

Figure 14: Comparative Overlay Size and Energy Data for Initial and Revised Prototypes

5.4 Considerations for Data-Reliant Applications

A large corpus of data is an important part of many applications that are relevant to hostile environments. For example, a face-recognition application typically requires a collection of facial images of persons of interest. Including this large data source as part of the overlay will increase its size, which in turn will require a greater transmission time, which will lead to greater battery consumption. This situation is unacceptable in many environments beyond hostile environments.

We implemented a solution by installing the face database in a Samba share inside the Cloudlet Host so that the database would be accessible from the Windows-based Face Recognition Server [Eckstein 1999]. During the "Install Application" step of the process, outlined in Figure 10, the database location (part of the application configuration file) was changed to point to the Samba share. An advantage of this approach is that the data is not part of the overlay. In addition, multiple VM instances running the Face Recognition Server could share the same database. The disadvantage is additional complexity of the cloudlet-deployment process, for two reasons:

1. Installation of the Samba share would have to become part of the "standard cloudlet configuration."
2. Databases for every potential application that would run on the cloudlet would require installation inside the Samba share, either at cloudlet-configuration time or at application-install time.

Another scenario to consider is one in which the data source is shared by many cloudlets, in which updates may be performed in the field at many different locations. VMs running the face-recognition application inside each cloudlet will need access to relevant parts of this data collection. These types of applications need cloud-wide distributed data access that is resilient to network disruptions. An option is to use a cloud-based distributed file system with a persistent cache of file data on each cloudlet that can work in disconnected mode. An example of such a system is *Coda*—a cloud-based distributed file system that features aggressive client caching on local disks, callback-based cache coherence, location transparency, organization of the file-name space into logical volumes to facilitate system administration, and file protection based on access-control lists [CMU 2012c]. We installed a Coda client on the Cloudlet Host within the Samba share and pointed the data location for the face-recognition application to the Coda cache. The Coda server that contained the database was on another machine that we called the cloud. We achieved the same results as when the data source resided on the cloudlet. The advantage is that all instances of the Face Recognition Server running in any cloudlet would have access to the same data. The cloudlet Coda cache would be updated as needed if there were access to the cloud. Otherwise the application would still work with outdated data until access to the cloud was restored. The disadvantage is that Coda would have to become part of the "standard cloudlet configuration," as in the previous case.

6 Related Work

There is a considerable amount of work, conducted as early as 2001, that relates to code offload from mobile devices to cloud environments [Balan 2002, 2007; Christensen 2009; de Lara 2001; de Leusse 2008; Flinn 2001, 2002; Goyal 2004; Kemp 2009; Kumar 2010; Li 2009; Marinelli 2009; Ok 2007; Palmer 2009; Satyanarayanan 2001; Zhang 2009]. However, this work in cyber-foraging from mobile devices assumes that acceptable networking conditions prevail between a mobile device and its offload site. Although the bandwidth and latency of this connectivity varied, a universal assumption in previous work was that connectivity was "good enough." To the best of our knowledge, our work is the first to investigate the challenges of cloud offload in hostile environments and to propose an architectural solution to the problem.

One example of closely related work is MAUI [Cuervo 2010]. MAUI is a system that enables the fine-grained energy-aware offload of mobile code to offload elements, with minimum programmer effort—code annotations indicate methods that could be executed remotely. However, this approach is platform specific (Microsoft .NET) and therefore would limit the applications that could be offloaded.

Another closely related effort is CloneCloud [Chun 2011]. Unlike with MAUI, applications do not require modification. Before execution, the partitioner portion of the system uses static analysis and dynamic profiling to partition the application executable, based on optimization of execution time and energy use. The runtime portion of the system enables the application, which is running in an application-level VM, to automatically migrate threads from the mobile device (based on the partitioning information) to a device clone running in the cloud and to reintegrate the migrated thread along with its state back to the mobile device when execution is finished. This work also assumes connection to the cloud, but it could be implemented such that threads are migrated to a cloudlet instead of a cloud. However, supported platforms are limited (e.g., Java VM, Android Dalvik VM, Microsoft .NET) and deployment and hardware requirements would be difficult to achieve in tactical-edge environments.

Other work that establishes a foundation for work in cyber-foraging includes the following:

- Goyal and Carter proposed a VM-based approach in which a discoverable virtual machine server acts as a surrogate to run client application code [Goyal 2004]. This approach addresses security issues, but requires the surrogate to be connected to the internet to locate and download client application code and also requires code to be partitioned at development time.
- Similar to this approach is the Locusts framework that enables the discovery of cyber-foraging resources (surrogate peers) [SourceForge.net 2009]. Peers can offload coarse-grained tasks to other peers as well as process offloaded tasks from other peers [Kristensen 2008]. Applications must be partitioned in advance into locally executed code and remotely executable tasks.
- Work proposed by Chen and colleagues is not VM-based, but includes a mechanism for deciding (1) whether to execute locally or remotely, based on size of method input data and

wireless channel conditions, and (2) whether to interpret bytecode or compile native code [Chen 2004]. The solution works only for Java code, and applications require modifying to enable code offload.

- Kemp and colleagues propose an approach that leverages the Ibis high-performance distributed-computing middleware [Kemp 2009]. The server portion is sent from the mobile device to the surrogate at runtime, but applications must be written as distributed applications using the Ibis programming environment and the server.

Our work implements an instance of cloudlet strategy presented by Satyanarayanan that uses a thick-client approach (native app) instead of a thin, VNC-based client [Satyanarayanan 2009]. The advantage of the VNC approach is that applications would not require any modification because they would run completely on the server. However, many of the applications at the edge use sensors to capture information about the environment. A better fit for the edge scenario involves splitting the application into (1) a very simple native app that runs on the mobile device to capture manual or sensed input, and (2) a server portion that runs all the expensive computation. The advantage of the VM-based approach is simplicity of setup and deployment that relies on (1) generic servers running pre-configured Base VMs (cloudlets) and (2) cloudlet-ready mobile devices loaded with overlays for computation-intensive applications. There is no need for special hardware or middleware.

7 Conclusions and Future Work

Today there is substantial consensus that cloud offload of resource-intensive application execution is a core technique in mobile computing. In this report, we have described a reference architecture for code offload in tactical-edge environments and presented two viable implementations along with their architectural tradeoffs. A difficult problem exposed by this architecture involves rapid delivery of large application overlays to offload sites as well as rapid application ready time.

Current and possible future research that we are exploring includes

- other forms of code offload, such as demand paging from the cloud, that consume less energy and provide better launch times but have reliable communication between cloudlets and the cloud [Ha 2011]
- rapid VM synthesis, through
  - extension of the discovery protocol to enable VM caching so that overlays do not always require transmission
  - exploiting of multicore architecture to parallelize VM synthesis activities
- mobility-induced cloudlet handoffs to transfer state between cloudlets with minimal interruption to a moving user
- a cloudlet-selection mechanism that maps application needs to cloudlet characteristics exposed as cloudlet metadata during the cloudlet-discovery process

If cyber-attacks become more prevalent on the public internet, cloud offload of mobile devices will become increasingly unreliable. Someday, the entire public internet may have to be viewed as a hostile environment. The issues explored in this report in the context of the tactical edge will then hold much broader relevance.

References/Bibliography

URLs are valid as of the publication date of this document.

[7Zip 2012]
7-Zip.org. LZMA SDK (Software Development Kit). http://www.7-zip.org/sdk.html (2012).

[Avahi 2012]
Avahi.org. Avahi. http://avahi.org/ (2012).

[Balan 2002]
Balan, R., Flinn, J., Satyanarayanan, M., Sinnamohideen, S., & Yang, H. "The Case for Cyber Foraging," 87-92. *Proceedings of the 10th ACM SIGOPS European Workshop*. Saint-Emilion, France, September 2002. ACM, 2002.

[Balan 2003]
Balan, R., Satyanarayanan, M., Okoshi, T., & Park, S. "Tactics-Based Remote Execution for Mobile Computing," 317-330. *Proceedings of the 1st International Conference on Mobile Systems, Applications and Services.* San Francisco, CA, May 2003. USENIX Association, 2003.

[Balan 2007]
Balan, R., Gergle, D., Satyanarayanan, M., & Herbsleb, J. "Simplifying Cyber Foraging for Mobile Devices," 272-285. *Proceedings of the 5th International Conference on Mobile Systems Applications and Services*. San Juan, Puerto Rico, June 2007. ACM, 2007.

[Chen 2004]
Chen, G., Kang, B., Kandemir, M., Vijaykrishnan, N., Irwin, M., & Chandramouli, R. "Studying Energy Trade Offs in Offloading Computation/Compilation in Java-Enabled Mobile Devices." *IEEE Transactions on Parallel and Distributed Systems*, *15*, 9 (September 2004): 795-809.

[CherryPy 2012]
CherryPy.org. Cherry Py—A Minimalist Python Web Framework. http://www.cherrypy.org/ (2012).

[Christensen 2009]
Christensen, J. "Using RESTful Web Services and Cloud Computing to Create Next-Generation Mobile Applications," 627-634. *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object-Oriented Programming Systems Languages and Applications (OOPSLA '09).* Orlando, FL, October 2009. ACM, 2009.

[Chun 2011]
Chun, B., Ihm, S., Maniatis, P., Naik, M., & Patti, A. "CloneCloud: Elastic Execution between Mobile Device and Cloud," 301-314. *EuroSys 2011*. Saltzburg, Austria, April 2011. ACM, 2011. http://berkeley.intel-research.net/maniatis/publications/2011EuroSys-CloneCloud.pdf

[CMU 2012a]
Carnegie Mellon University Personal Robotics Lab. *MOPED: Object Recognition and Pose Estimation for Manipulation.* http://personalrobotics.ri.cmu.edu/projects/moped.php (2012).

[CMU 2012b]
Carnegie Mellon University. Sphinx-4: A Speech Recognizer Written Entirely in the Java Programming Language. http://cmusphinx.sourceforge.net/sphinx4/ (2012).

[CMU 2012c]
Carnegie Mellon University. Coda File System. http://coda.cs.cmu.edu (2012).

[Cuervo 2010]
Cuervo, E., Balasubramanian, A., Cho, D., Wolman, A., Saroiu, S., Chandra, R., & Bahl, P. "MAUI: Making Smartphones Last Longer with Code Offload," 49-72. *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys '10*). San Francisco, CA. June 2010. ACM, 2010.

[de Lara 2001]
de Lara, E., Wallach, D. S., & Zwaenepoel, W. "Puppeteer: Component-based Adaptation for Mobile Computing," 14-14. *Proceedings of the 3rd Conference on USENIX Symposium on Internet Technologies and Systems (USITS '01).* San Francisco, CA, March 2001. USENIX Association, 2001.

[de Leusse 2008]
de Leusse, P., Periorellis, P., Watson, P., & Maierhofer, A. "Secure and Rapid Composition of Infrastructure Services in the Cloud," 770-775. *Proceedings of the 2nd International Conference on Sensor Technologies and Applications*. Cap Esterel, France. August, 2008. IEEE, 2008.

[Dilger 2011]
Dilger, D. "First Look: Using iPhone 4S with Siri Voice Assistant (with videos)." Apple Insider. http://www.appleinsider.com/articles/11/10/14/first_look_using_iphone_4s_with_siri_voice_assistant.html (2011).

[Eckstein 1999]
Eckstein, R., Collier-Brown, D., & Kelly, P. *Using Samba*. O'Reilly, Inc., November 1999.

[Flinn 2001]
Flinn, J., Narayanan, D., & Satyanarayanan, M. "Self-Tuned Remote Execution for Pervasive Computing," 61-66. *Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems*. Schloss Elmau, Germany, May 2001. IEEE, 2001.

[Flinn 2002]
Flinn, J., Park, S., & Satyanarayanan, M. "Balancing Performance, Energy Conservation and Application Quality in Pervasive Computing," 217-226. *Proceedings of the 22nd International Conference on Distributed Computing Systems.* Vienna, Austria, July 2002. IEEE Computer Society, 2002.

[GitHub 2012]
Github, Inc. seb-m/pyinotify - Github. https://github.com/seb-m/pyinotify (2012).

[Goyal 2004]
Goyal, S. & Carter, J. "A Lightweight Secure Cyber Foraging Infrastructure for Resource-Constrained Devices," 186-195. *Proceedings of the 6th IEEE Workshop on Mobile Computing Systems and Applications.* Lake District National Park, UK, December 2004. IEEE Computer Society.

[Ha 2011]
Ha, K., Lewis, G., Simanta, S., & Satyanarayanan, M. *Code Offload in Hostile Environments (CMU-CS-11-146).* Carnegie Mellon University. 2011. http://reports-archive.adm.cs.cmu.edu/anon/2011/CMU-CS-11-146.pdf

[Kemp 2009]
Kemp, R., Palmer, N., Kielmann, T., Seinstra, F., Drost, N., Maassen, J., & Bal, H. "eyeDentify: Multimedia Cyber Foraging from a Smartphone," 392-399. *Proceedings of the 11th IEEE International Symposium on Multimedia*. San Diego, CA, December 2009. IEEE Computer Society, 2009.

[Kristensen 2008]
Kristensen, M. D. "Execution Plans for Cyber Foraging," 1-6. *MobMid '08: Proceedings of the 1st Workshop on Mobile Middleware.* Leuven, Belgium, December 2008. ACM, 2008.

[Kumar 2010]
Kumar, K. & Lu, Y. "Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?" *IEEE Computer 43*, (April 2010): 51-56.

[KVM 2012a]
KVM. Kernel-Based Virtual Machine. http://www.linux-kvm.org/ (2012).

[KVM 2012b]
KVM. Guest Support Status. http://www.linux-kvm.org/page/Guest_Support_Status (2012).

[Lehr 2002]
Lehr, W. & McKnight, L. Wireless Internet Access: 3G vs. WiFi? Center for eBusiness @ MIT. 2002. http://www.sciencedirect.com/science/article/pii/S0308596103000041

[Li 2009]
Li, X., Zhang, H., & Zhang, Y. "Deploying Mobile Computation in Cloud Service," 301-311. *Proceedings of the 1st International Conference on Cloud Computing (CloudCom '09).* Bejing, China, December 2009. Springer-Verlag, 2009.

[Marinelli 2009]
Marinelli, E. Hyrax: *Cloud Computing on Mobile Devices using MapReduce (CMU-CS-09-164).* Carnegie Mellon University, 2009. http://reports-archive.adm.cs.cmu.edu/anon/2009/CMU-CS-09-164.pdf

[Microsoft 2011]
Microsoft Corporation. TCP/IP and NBT configuration parameters for Windows XP. http://support.microsoft.com/kb/314053 (2011).

[Monsoon 2012]
Monsoon Solutions, Inc. Power Monitor. http://www.msoon.com/LabEquipment/PowerMonitor/ (2012).

[Morris 2011]
Morris, Edwin. "A New Approach for Handheld Devices in the Military." SEI Blog. http://blog.sei.cmu.edu/post.cfm/a-new-approach-for-handheld-devices-in-the-military (2011).

[Ok 2007]
Ok, M., Seo, J.-W., & Park, M.-S. "A Distributed Resource Furnishing to Offload Resource-Constrained Devices in Cyber Foraging Toward Pervasive Computing." T. Enokido, L. Barolli, & M. Takizawa, Eds, "Network-Based Information Systems." *Lecture Notes in Computer Science 4658,* Springer Berlin / Heidelberg, 2007.

[OpenCV 2012]
OpenCV. Welcome – Open CV Wiki. http://opencv.willowgarage.com/wiki/ (2012).

[Palmer 2009]
Palmer, N., Kemp, R., Kielmann, T., & Ba, H. "Ibis for Mobility: Solving Challenges of Mobile Computing Using Grid Techniques." *Proceedings of the 10th Workshop on Mobile Computing Systems and Applications*. Santa Cruz, CA, February 2009. ACM, 2009.

[Richardson 1998]
Richardson, Tristan, Stafford-Fraser, Quentin, Wood, Kenneth R., & Hopper, Andy. "Virtual Network Computing." *IEEE Internet Computing 2*, 1 (January 1998): 33-38. DOI=10.1109/4236.656066 http://dx.doi.org/10.1109/4236.656066

[Satyanarayanan 1996]
Satyanarayanan, M. "Fundamental Challenges in Mobile Computing," 1-7. *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing (PODC '96*). Philadephia, PA, May 1996. ACM, 1996.

[Satyanarayanan 2001]
Satyanarayanan, M. "Pervasive Computing: Vision and Challenges," 10-17. *IEEE Personal Communications 8,* 4 (August 2001). IEEE, 2001.

[Satyanarayanan 2009]
Satyanarayanan, M., Bahl, P., Caceres R., & Davies N. "The Case for VM-Based Cloudlets in Mobile Computing." *IEEE CS Pervasive Computing 8*, 4 (November–December 2009): 14-23.

[SourceForge.net 2009]
SourceForget.net Locust Framework. http://locustframework.sourceforge.net/ (2009).

[Ubuntu 2012a]
Ubuntu. KVM/Networking – Community Ubuntu Documentation. https://help.ubuntu.com/community/KVM/Networking (2012).

[Ubuntu 2012b]
Ubuntu. Samba File Server. https://help.ubuntu.com/11.04/serverguide/C/samba-fileserver.html (2012).

[Wolbach 2008]
Wolbach, A. *Improving the Deployability of Diamond (CMU-CS-08-158).* Carnegie Mellon University School of Computer Science. 2008. http://reports-archive.adm.cs.cmu.edu/anon/anon/2008/CMU-CS-08-158.pdf

[xdelta 2012]
xdelta.org. xdelta. http://xdelta.org (2012).

[Zeroconf 2012]
Zero Configuration Networking (Zeroconf). http://www.zeroconf.org (2012).

[Zhang 2009]
Zhang, X., Schiffman, J., Gibbs, S., Kunjithapatham, A., & Jeong, S. "Securing Elastic Applications on Mobile Devices for Cloud Computing," 127-134. *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*. Chicago, IL, November 2009.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	October 2012	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Cloud Computing at the Tactical Edge	FA8721-05-C-0003

6. AUTHOR(S)
Soumya Simanta, Grace A. Lewis, Edwin Morris, Kiryong Ha, Mahadev Satyanarayanan

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2012-TN-015

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
ESC/CAA 20 Schilling Circle, Building 1305, 3rd Floor Hanscom AFB, MA 01731-2125	n/a

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

Handheld mobile technology is reaching first responders, disaster relief workers, and soldiers in the field to aid in various tasks, such as speech and image recognition, natural language processing, decision making, and mission planning. However, these applications are computation intensive, so it is necessary to consider that (1) mobile devices offer less computational power than conventional desktop or server computers, (2) computation-intensive tasks consume large amounts of battery power, and (3) networks in hostile environments, such as those experienced by first responders and soldiers in the field, are often unreliable, and bandwidth is limited and inconsistent. While there has been considerable research in code offload to the cloud to enhance computation and battery life, most of this work assumes reliable connectivity between the mobile device and the cloud—an invalid assumption in hostile environments. This technical note presents a reference architecture for mobile devices that exploits cloudlets—virtual-machine-based, code-offload elements—that are in single-hop proximity to the mobile devices that they serve. Two implementations of this reference architecture are presented, along with an analysis of architecture tradeoffs.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Cloudlet, face recognition, virtual machine, architecture	41

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL